Architectural design decisions

Jansen, Antonius Gradus Johannes

CHAPTER 8

CONCLUSIONS

This chapter presents the conclusions of this thesis. First, the answers are presented to the research questions underlying this thesis. This is followed by a description of the contributions this thesis makes to the software engineering field. The chapter concludes with an outlook into future research directions and open research questions.

8.1 Research Questions & Answers

In section 1.5 on page 16, the research questions of this thesis were presented. For each of these questions, we present a short answer here based on the work presented in the previous chapters. We start with the detailed research questions and conclude with answering the two more general research questions.

RQ-1: What are architectural design decisions?

We have provided a definition (see sections 3.4.3, 4.2, 6.2) and a conceptual model (see sections 3.5.1, 4.4.1, 7.2.2) to answer this question. There are two interpretations of this concept: a narrow and a broad one. In the narrow interpretation, an architectural decision is a *choice* among several architectural alternatives. This thesis, however, follows the broader interpretation in which architectural design decisions are much more than this choice alone. The conceptual model of section 7.2.2 describes this broader interpretation. Besides the choice it defines the following concepts to be part of an architectural design decision:

- **Problem,** a way to scope the problem space.
- **Motivation,** the rationale of this scoping.
- **Cause,** an analysis of the things that have lead to this scoping and motivation.
- **Solutions,** the alternatives considered for the choice along with their pros, cons, rules, and constraints.
- **Trade-offs,** the balances offered by the various solutions among quality attributes and functionality.
- **Architectural modification,** the effect the choice has on the software architecture.

RQ-2.1: How can we model features in an SPL?

Features in an SPL can be seen as a special type of architectural design decisions (see 1.3.3). Before answering the question how architectural design decisions can be modeled (RQ-2), we answer this question for this specific type of architectural design decisions. In chapter 2, we presented a role based feature model for modeling features in an SPL. The influence a feature might have on a component is isolated in a role. In this way, features can be described independently from each other. However, this separation of concerns comes at the price of making it hard to combine individual features. The complicating factors are feature interactions and the dependencies they represent, which conflict with the aim of describing features as separate concerns. How to deal with these feature interactions is the topic of RQ-3.1. Similar to this, a research question has been posed how to deal with architectural design decision dependencies (RQ-3).

RQ-2: How can we model architectural design decisions?

The Archium meta-model (see A.1 on page 195) presented in chapters 4 and 5 models architectural (design) decisions. The meta-model models the elements described in the conceptual model for architectural design decisions (see RQ-1), which is based on the broad definition of this concept.

Similar to the way the role based feature model tries to make features first class citizens (see chapter 2), the Archium meta-model tries to achieve this for architectural design decisions. In particular, the modeling of the architectural modification, i.e. the changes to the architecture design of an architectural design decision, is inspired by the feature model. The concept of a delta comes from the concept of a role from the feature model. The delta concept allows Archium to separate the architectural modification from the architectural design. Consequently, this supports the modeling of architectural design decisions as first class citizens in Archium.

Archium adopts the broad definition of architectural design decisions (see the answer to RQ-1). This implies that a rich context is modeled in the Archium meta-model. To this end, the meta-model uses five (sub)models, which not only model architectural design decisions, but their context as well. The five (sub)models the Archium meta-model consists of are the following:

- **Requirements model,** for modeling the concerns that come from various stakeholders, which influence architectural decisions.
- **Decision model,** for modeling the architectural design decisions themselves.
- **Architecture model,** for modeling the effects an architectural design decision has on the architecture.
- **Implementation model,** for modeling how the architecture relates to the implementation. In the case of the Archium tool, the Java programming language is used.
- **Composition model,** is used for combining the decision, architecture, and implementation models.

RQ-3.1:How to deal with feature interactions?

One way to deal with feature interactions is to use a flexible composition approach with sufficient expressive power. This holds for situations in which features are modelled in isolation, such as done in the role based feature model presented in chapter 2 (see also RQ-2.1). In this situation, feature interactions and the related problems become apparent when multiple features are combined during composition.

To resolve these problems, a powerful and flexible composition approach is needed. To have enough expression power, the approach should support the three basic solutions for dealing with composition problems: skipping, mixing, and concatenation (see chapter 2). In addition, it should have the flexibility to combine one or more of these solutions at different levels of abstraction. The composition model of the role based feature model tries to achieve these two aspects. Although it supports the three basic composition solutions, it is rather limited in terms of flexibility. Nevertheless, it still provides a powerful and sensible approach to deal with feature interactions.

RQ-3:How to deal with architectural design decision dependencies?

To deal with architectural design decision dependencies, we provide a solution similar to the separation of concerns approach used for feature interactions. The

Archium meta-model (see chapter 4 on page 79 and A.1 on page 195) models architectural design decisions using a similar separation-of-concerns modeling approach to the one used for the role based feature model (see chapter 2). Thus, Archium suffers from at least the same kind of composition problems as the feature model, since the dependencies among architectural design decisions might be more extensive than for features in an SPL. To what extent this is the case is an open research question.

The composition model of Archium uses superimposition as its composition technique, thereby offering support for the three basic composition solutions (see chapter 2). In comparison with the feature model, Archium offers a lot more flexibility than the inheritance based composition technique used in the role based feature model. The basic composition solutions can be freely combined through the use of superimposition [18]. In addition, Archium offers a much wider range of granularity for the composition, which ranges from a single method invocation to an entire software architecture design.

Although the composition model of Archium provides a way to deal with architectural design decisions dependencies, the approach does not model these dependencies explicitly. Instead, Archium infers the dependencies between architectural design decisions. This inference is based on found composition dependencies and references in the textual elements of an architectural design decision to other decisions. Complicating matters is the run-time binding of architectural design decisions in the Archium tool. This late binding time approach also delays the verification of these implicit dependencies. In many cases, it would be more convenient if these dependencies could be verified at design or compile time, as to warn the user of potential problems in advance. However, this would come at the price of reducing the flexibility of the composition of the approach. The best approach is most likely a hybrid approach, in which an architect can choose the binding time herself. In conclusion, Archium provides one way to deal with the architectural design decision dependencies, but there is room for improvement, i.e. the verification of dependencies in advance.

RQ-4: What is the added value of explicit architectural decisions in the architecting process?

Explicit architectural decisions improve the architecting process in various ways. Among others, they enable architects to make better architectural choices. They also improve communication about the software architecture among the various stakeholders. In addition, they allow stakeholders to evaluate various software ar-

chitecture options better. This is due to the following benefits of the concept of explicit architectural decisions (see also sections 3.3.2, 4.3, and 7.2):

- **Focus** the architecting process by explicit scoping and selecting a subset of the problems.
- **Guarding of conceptual integrity** as rules, constraints, and intent of previous decisions are known.
- **Improved understandability** as it provides a rationale for the choices made.
- **Explicit design space exploration** as it charts the explored design space by describing the alternative solutions considered. This prevents reconsideration of earlier rejected solutions and eases the identification of unexplored areas.
- **Support of trade-off reconsideration** as explicit architectural design decisions allow for revisiting those decisions that, in retrospect, involve trade-offs that are problematic. A first benefit is that the effort to locate these design decisions is reduced significantly. Second, as these decisions describe the considered alternatives, the effort to think up suitable alternatives is reduced. Third, the impact of changing a decision can be predicted [152] and a rough estimate can be made which part of the system should be redesigned.
- **Decision analysis support** as explicit decisions allow one to easily perform what-if scenarios and thus identify which decisions are important sensitivity and trade-off points.
- **Tracing support** among architectural elements and requirements. For example, tracing the impact a design decision has on various views of the software architecture provides a mechanism to relate the elements of different views.

RQ-5: How is making architectural decisions part of the architecting process?

The decision process of making architectural decisions is the driving force behind the architecting process. Both the decision circle (see section 7.2 on page 145) and the similarities between the architecting process with the rationale management process (see section 3.4 on page 64) illustrate this. Architectural analysis, synthesis, and evaluation are all activities for the purpose of making architectural decisions. Hence, the concept of architectural decisions is omnipresent in the architecting process. Consequently, there is not a single place in the architecting process where architectural decisions are made; rather, they drive the process in the background.

RQ-6: What are existing automated support tools for managing architectural design decisions and what do they support?

In this thesis, we addressed this research question in the context of architectural evolution. For a knowledge sharing perspective on this issues we refer to [46]. Rationale management tools (e.g. Compendium) provide support for explicit decision making. However, they do not support the notion of a software architecture, nor are they suited for coping with changes induced by architectural evolution. ADLs and component languages do. on the other hand, support the description of software architectures to a certain degree, but lack the notion of architectural (design) decisions. In addition, both ADLs and component languages (apart from SOFA [121]) have sub-optimal support for changes induced by architectural evolution. Consequently, there exists a gap between these tools and decision management tools. The evaluation of the Archium tool (see 6.4.6) demonstrates that it can bridge this gap. Nevertheless, there is room for improvement in the areas of support for architectural views and support for the refinement process. Archium does, however, address the issue of changes induced by architectural evolution, something the other evaluated tools have trouble with.

RQ-7: How to provide good tool support for architectural decisions?

We have identified 27 use cases involving AK, based on interviews with software architects and project managers from industry [162]. For the nine most important use cases, we explain how the Archium tool can support them (see chapter 5). Thus, we present a way in which partial tool support for architectural decisions could be provided.

RQ-8: How can we recover architectural design decisions?

The Architecture Design Decision Recovery Approach (ADDRA) (see chapter 7) outlines the steps and techniques one can use to recover architectural decisions. The approach uses a combination of architectural recovery and knowledge externalization techniques. The tacit knowledge of the original architect plays a crucial role in this kind of recovery, as the choices for abstraction of the architectural design decisions are otherwise unrecoverable. We present various ways in which this dependency on the original architect can be minimized.

RQ: How to reduce the vaporization of architectural decisions?

The answer to this question depends on the knowledge management strategy of an organization (see also section 1.3.1). In this thesis, the focus was on organizations using an explicit codification strategy. To eliminate architectural decision vaporization in this case, we need to create, manage, and maintain explicit architectural decisions throughout the life-cycle. Achieving this is far from trivial and also expensive, as architectural decisions are typically intertwined with each other and cross-cut the software architecture.

For forward engineering purposes, this thesis presents Archium, which addresses the two aforementioned issues of intertwining and cross-cutting. In practice, however, creating explicit architectural decisions is not always feasible. Thus, for reverse engineering purposes we presented ADDRA to recover these decisions. Together, Archium and ADDRA help the architect to create, manage, and maintain explicit architectural design decisions during the life-cycle.

How to reduce the vaporization of architectural knowledge?

This thesis presented a small step towards an answer to this research question. Our focus was primarily on one type of AK: architectural design decisions. In addition, we focussed exclusively on finding solutions in the context of a codification knowledge management strategy. To provide a general answer to this research question, we need to investigate other kinds of AK (e.g. process and people related) for both the codification and personalization strategies.

8.2 Contributions

The previous section already presented some of the contributions of this thesis. In this section, we present an overview of them. In short, this thesis made the following contributions:

- **Role-based SPF feature model** Chapter 2 presented a role-based approach for modeling features in a SPF. This model allows for automatic derivation of products based on feature selections. In general, feature models for product derivation suffer from the feature interaction problem. We analyzed this problem and concluded that, for automatic product derivation, feature interactions boil down to composition problems. Three basic solutions to these composition problems have been identified: concatenation, skipping, and mixing. The use of one or

more of these solutions allows for the automatic derivation of products based on features in an SPF. The insights gained from this model are used in the Archium meta-model and tool.

- **The bridging function of design decisions** We presented in chapter 3 how the concept of (architectural) design decisions forms a bridge between rationale and software architecture. To understand this bridging function, we analyzed the software architecture design process. This process assumes that the software architecture is designed in iterations. In each iteration, an architectural modification is made and incorporated into the software architecture. This is followed by an assessment to determine whether the quality of the software architecture is sufficient and thus to determine whether more iterations are needed. Design decisions come into this process through the architectural modifications, as they capture both the rationale behind these modifications and the considered alternatives. Consequently, design decisions form a bridge between rationale and software architecture.
- **A conceptual model for architectural design decisions** The conceptual model (see sections 3.5.1, 4.4.1, 7.2.2) of architectural design decisions describes what this concept entails. Furthermore, it describes the concepts of the context in which the decisions are made. The conceptual model can be used as a basis for templates that describe architectural design decisions. For example, in chapter 7, this is done for recovered architectural design decisions. The conceptual model is also used as a conceptual basis for making architectural design decisions explicit. The Archium meta-model uses the conceptual model towards this purpose.
- **Archium meta-model** The Archium meta-model, presented in chapters 4 and 5, models how architectural design decisions can be given a first-class representation (i.e. an identifiable nameable object) in architectural design. It incorporates the aforementioned conceptual model to model architectural (design) decisions. It also models the context of these decisions in more detail. The meta-model describes the relationships various concepts have with architectural (design) decisions and defines semantics for these relationships. For example, the relationships that architectural decisions have with requirements or components.

  A novel perspective incorporated in the meta-model is to model a software architecture design as a set of architectural (design) decisions. To achieve this, the meta-model uses, among others, two new concepts; deltas and design fragments. The delta concept originates from insights gained from the role based feature model. The concept of a delta models the influence an architectural design decision has on a component. It has been inspired by the feature role

concept. The second novel concept, is the design fragment. This concept represents a piece of architecture, thereby making it possible to model concepts like architectural design decisions that work on parts of the architecture.

- **Archium tool** The Archium tool described in chapters 4 and 5 is one way to implement the Archium meta-model. The tool enables architects to create, manage, and use architectural (design) decisions. As its implementation model, the tool uses the Java language. This offers the architect a way to maintain a software architecture and its implementation, including the architectural design decisions. This can be done not only during the design phase, but also during the remaining part of the system life-cycle, as the decisions become an inherent and explicit part of the system.
- **New views and notations** To communicate the different concepts of the Archium meta-model, we have developed various views and notations. For example, figure 4.4 on page 91 graphically displays design fragment composition. Another example is figure 4.6 on page 95, which presents an evolutionary view on architectural design decisions. The Archium tool also offers a design decision impact view by combining a design decision dependency view and a component & connector view (see figure 5.1 on page 109).
- **Evaluation of tool support for architectural evolution** Chapter 6 presents a framework for the evaluation of tool support for architectural evolution. It allows one to judge tools with respect to their ability to support software architecture evolution. The framework consists of coarse grained criteria concerning the architecture, architectural design decisions, and architectural changes for evaluation. We have applied the evaluation framework on six different tools. From the results, we conclude that, in general, there exists a gap between software architecture tools on one side and knowledge management tools on the other side. The Archium tool bridges this gap, but can be improved in the area of support for multiple architectural views and facilities for expressing incompleteness by under-specification.
- **ADDRA** The Architectural Design Decision Recovery Approach (ADDRA) approach presented in chapter 7 offers the architect a means to recover architectural design decisions. The approach supports the recovery of decisions months or even years after they have been made. To this end, It uses a combination of architectural recovery techniques, differences in recovered architectures, and knowledge externalization strategies.

8.3 Open research questions and future work

Not all the research questions of this thesis were completely answered in section 8.1. In this section, we present these research questions that are still partially open together with possible directions for future work.

This thesis focussed on organizations using a codification knowledge management strategy for minimizing architectural knowledge vaporization. However, many organizations do not use a codification strategy, but use a personalization strategy. An open research questions is therefore how to minimize architectural knowledge vaporization in organizations using a personalization knowledge management strategy.

The idea of viewing design patterns as collections of reuseable design decisions is another direction for future work. For example, Harrison et al [64] explored how patterns can be used to capture architectural design decisions. Another interesting direction is to investigate the kind of design decisions made to integrate a pattern in a software architecture. This might provide the basis for offering better (tool) support for these kind of re-usable design decisions.

An other interesting open question is how architectural design decisions can be undone. Undoing architectural design decisions is a special case of modifying the architecture with the aim to remove constraints, rules, design modifications, or unwanted dependencies imposed by a decision. Of interest is undoing these decisions not only during design, but also during later phases of the life cycle. For example, in Archium, one could investigate how architectural design decisions could be reverted at run-time, including the implications this brings for the approach.

Another direction for future work is to look into the relationship an Archium like tool might have with software architecture documentation. If at all, formalization of the architecture, as supported by Archium, is usually done after a software architecture is textually described. Investigating how such a software architecture document could be refined and later transformed into a formal model might provide some insights into the problems with formalizing an architectural description. This knowledge in turn could be used to optimize the translation from an informal architecture description to a formal one.

The research question about how to deal with architectural design decision dependencies (RQ-3) was only partially answered. The answer from the role based feature model (see chapter 2) is incomplete and focusses on the role these dependencies play in a composition. However, the role the dependencies that are represented by textual references in Archium actually play are not dealt with. One direction for

future work is therefore to investigate the semantics of these textual relationships and the concepts they represent.

So far Archium has only been applied to small case studies. To further validate the Archium approach, bigger case studies are necessary. This comes not without challenges, as evolution forms an inherent part of the approach. Thus a part of the evolution of such a large case should be tracked, which complicates things and requires considerable effort.

The Archium tool is in many aspects a research prototype and needs to be matured. Especially, some of the technical limitations the underlying ArchJava language imposes (e.g. no support for run-time decoupling of connectors) should be resolved. Hence, maturing the Archium tool is another challenge. Another direction for further work is to investigate how the Archium meta-model could be expanded to include more architectural views (e.g. a module or deployment view) besides the component & connector view.

In chapter 6, various tools were evaluated with respect to the support they could offer for the evolution of software architectures and architectural (design) decisions. However, tools for managing AK should be evaluated from other perspectives as well, e.g. for sharing [46], creating, and evaluating AK. In addition, several new tools for managing AK have been published, e.g. AREL [151], PAKME [6], ADDSS [26], after the evaluation. This is another direction for future work, which at this time of writing is being pursued actively by the authors of these tools.

For the recovery of architectural design decisions, the ADDRA approach has been validated using a case study. This proved that it is a possible to recover some architectural design decisions afterwards. However, to what extent this is possible is still an open research question (see also 7.8.2). Another open question is whether such a recovery makes economical sense, i.e. the benefits outweigh the cost of recovery. Thus these two questions provide another direction for further work.